A Framework for Private Location-based Queries using Cryptographic Protocols

Yan Huang, Stephen R. Tate, Roopa Vishwanathan

Abstract—An important privacy issue in Location Based Services (LBS) is to hide a user’s identity and location while still providing quality service. A user’s identity can be easily hidden through anonymous web browsing services, however, a user’s location can reveal a user’s identity.

For example, if a user at home asks queries such as “Find the nearest hospital around me” to a LBS server, then based on publically available demographic data and the location of the user, the server could potentially guess the identity of the querying user. A common way to achieve location privacy is through cloaking, e.g., the user/client sends a cloaked region to the server and filters the results to find the exact answer, but existing solutions such as cloaking using k-anonymity, or using a trusted anonymizer, aren’t completely privacy-preserving and leak some spatial information. In this paper, we propose an efficient two-phase framework for privacy in LBS based on two cryptographic protocols: Private Information Retrieval (PIR) and Oblivious Transfer (OT). Our framework provides privacy for the client and server, does not use a trusted party or anonymizer and is provably privacy-preserving. When compared to previous approaches, it ensures that the server reveals only as much data as is required by the querying client. In addition to providing a proof of security, we conduct experiments to measure the performance of our proposed framework and find that the costs of our solutions are reasonable and not prohibitively expensive to implement in practice.

1 INTRODUCTION

Location based services are becoming increasingly common with location-enabled client devices like mobile phones or PDAs making queries to location servers. Clients may not want to reveal their own identity and location while querying the server. A client’s identity can be hidden by using a fake one, but their location is needed to answer location-based queries. One way for the client to hide its own location is to query a large region and filter the query results to find its exact location within that region. As a result the server needs to respond to the clients’ queries with more data than what is needed. However, the server may not want to reveal unnecessary information that is not relevant to or close to the client’s precise location. Hence, we need to achieve a trade-off between the privacy of the client and the amount of data supplied by the server. Most solutions proposed till now take into account the privacy of the client. In addition to this, we also need to ensure that while protecting the privacy and anonymity of the client, we do not inadvertently release more information than what is required about other data and locations stored in the server. Typically a location server might contain thousands of points of interest (POIs) listed in a particular category (e.g., all restaurants in a given area) organized into a table, and if a user queries this table, the server does not want to return every item in its listing. In addition to being unnecessary, this would also enable a user to engage the server’s resources for long periods of time. Also, the location database is a potentially valuable asset for the server and if the server is operating on a commercial basis, it might be charging the client for each query; revealing extra information may not be in the business interests of the server. Hence we need to ensure that the data returned is fine-grained and precise (very close to the client’s location). This would also save the client time, since it does not have to sort through reams of extraneous data. That is, if the server maintains a number of POIs with each type organized into a table, it may allow clients to query part of the table about a limited region around a specific location, but not access all (or large amounts of) the data stored in it. For example, the server may allow queries like: “How many people are within 1 mile of me,” or “Where is the nearest gas station from me?” But the server may not want the client to ask queries along the lines of “Give me a list of all restaurants in the state of Texas”. One of the common queries in location services is a client query about their nearest neighbour(s). A client might query the location server to find its approximate nearest neighbour, exact nearest neighbour, k nearest neighbours, or reverse nearest neighbour. In this paper, we present a solution for the exact nearest neighbour query. Specifically, if a client/user $u$ has a location $l$ and needs to query a LBS to obtain its nearest neighbour from a set of POIs $S$ where $S = \{s_1, s_2, \ldots, s_o\}$, the goal is to develop an efficient protocol that does not allow the LBS to learn $l$ while reducing the subset of $S$ to be sent to $u$ in order to answer the query.

2 RELATED WORK

One line of previous work in privacy in location services mainly focuses on cloaking techniques using anonymizers [4],
In this approach, service requests are first transmitted to a third-party trusted server called location anonymizing server (AS). The AS strips off a service requester’s identifying information, such as user’s network address and name. Various perturbation operations can be performed on the AS to obfuscate the original location information. Location based service (LBS) requests are then issued from the AS to the LBS providers. The AS can also help filter the query results and return the exact answers to the original requesters. One major problem with this approach is that the anonymizer becomes a single point of failure and a performance bottleneck, and the client and server will have to trust the anonymizer.

Another solution is to organize the network as a peer-to-peer network, instead of having a centralized location server [3], [7]. In this approach, a mobile client first finds a peer group that meets the privacy requirement (i.e., finds $k-1$ neighbours) through peer searching. Then the client calculates a region that includes the $k-1$ peers (called $k$-anonymity [10]). It randomly chooses one of its neighbours as the agent to request LBS using the cloaking region. The query results are eventually forwarded to the original client through the agent. Since the disclosed location of a requester is expanded to an area that includes at least $k-1$ other mobile users, any of the $k$ people within the disclosed area could have been the user. For private locations that are likely to release a requester’s identification, the cloaking area is generally large if we use $k$-anonymity due to the small number of users in private (e.g., residential) areas. This approach requires the construction of a large peer network of mutually trusting and honest users and cannot guarantee service availability when clients are sparsely distributed.

Taking a spatial transformation-based approach Yiu et al. [29] develop a solution for preserving privacy in outsourced spatial data; this work is orthogonal to ours since it tackles a different problem, i.e., problems in a situation where the server outsources its spatial database to a third party, but transforms the database beforehand and shares the transformation key with the client(s). This methodology is more suitable for strengthening security in the $k$-anonymity approach (as the authors point out). Also, this work is vulnerable to access pattern or correlation attacks.

In work that uses cryptography for constructing privacy-preserving solutions, Wong et al. [28] present a way to securely compute $k$-nearest neighbors on an encrypted database. In their work, one requires $\Theta(n^2)$ time for each encryption and decryption operation, where $n$ is the number of elements in the database. Also [28] only focuses on server (database) privacy and does not consider the situation where the database tries to gain access to a querying client’s information. Additionally, [28] recommends using 80 different data dimensions for optimum security; one resulting problem of this approach is that there might not be enough indices for all the data points in the resulting space due to the curse of dimensionality [26].

Ghinita et al. [6] present a solution based on computational Private Information Retrieval (PIR) [2]. This approach has several advantages: it does not reveal any spatial information, it is resistant against correlation attacks, and it does not require any trusted third party, among others. The paper first presents a PIR-based framework for location based services and then presents solutions to the problems of finding approximate nearest neighbours and exact nearest neighbours. The approach is expensive in terms of communication and computation; some of the improvements and optimizations the authors have proposed include compressing the data sent by the server, having rectangular grids/matrices in addition to square ones, and using off-line data mining to avoid redundant computation. However the server reveals large amounts of extra information which results in exposing large portions of the server’s data assets to the client.

In addition to computational PIR as used by [6], there has been work in the area of hardware-assisted PIR and applying hardware-assisted PIR solutions to solve nearest neighbor privacy problems. In [27], Williams and Sion propose a hardware-assisted PIR scheme for obtaining practical privacy solutions. In their solution, clients need $O(\sqrt{n})$ working memory and they use a system augmented with a fairly expensive (upwards of $\$10,000$) secure co-processor. In our solution, clients do not need any extra working memory and do not need any physical enhancements to current business-class systems. In recent work that uses the hardware-assisted PIR protocol of [27], Papadopoulos et al. [24], present a framework for arbitrary $k$-nearest neighbor queries. We note that besides having the same issues as [27] (namely using an expensive, physical add-on co-processor and clients requiring working memory in the order of $O(\sqrt{n})$), [24] also leaks valuable database indexing information.

Along similar lines, recent work by Khoshgozaran et al. [18], [17] presents a solution where a powerful secure co-processor, SC, is placed close to (or inside) an untrusted LBS server, $LBS_DB$. The SC first shuffles the location database, $LBS_DB$ using a private shuffling function $\pi$ such that $LBS_DB[i] = LBS_DB[\pi(i)]$ and encrypts the shuffled database which can then be safely stored on the untrusted server. A querying client which wants to retrieve the $i^{th}$ bit from $LBS_DB$ then sends its query, $q = LBS_DB[i]$ encrypted with the public key of the SC to the server which passes it along to the SC. The SC decrypts $q$ to find $i$ and retrieves $LBS_DB[\pi(i)]$ from the untrusted server, decrypts it and again re-encrypts it with the client’s public key and sends it back to the client. The SC processes and returns answers to all client queries and through the entire process, the server remains oblivious as to what data the client has retrieved from it. Hengartner takes a trusted computing-based approach in [12], in which the execution of the PIR protocol is done by and responses to queries (locations) are given by a trusted module such as a Trusted Platform Module (TPM) chip. This methodology has not been implemented yet. Other work by Khoshgozaran and Shahabi on nearest neighbour queries uses tamper-evident hardware on both the server and client side [16]. The basic idea is to utilize one-way transformations to map the space of all static and dynamic objects to another space and resolve the query blindly in the transformed space. This approach requires that the transformation used preserves relative proximity of POIs. In another related paper Ghinita et al. [5] present a hybrid technique that combines cloaking and PIR, but the resulting solution reveals coarse-grained information about a client’s location to the server.
Hardware-assisted PIR, in general, requires current systems to be augmented with powerful, expensive secure co-processors (such as the IBM 4764) and cannot be realized as is with existing systems, either with no co-processors required, or by using an inexpensive, widely deployed commodity processor such as RSA’s SecureID tokens [20], Verisign’s e-token [14], Entrust’s IdentityGuard minitoken [13] or the Trusted Computing Group’s Trusted Platform Module (TPM) [9] chip. It would be ideal to realize hardware-assisted PIR using a cost-effective, already widely deployed hardware token, but such tokens are necessarily computationally lightweight, in order to keep their costs down and to encourage their widespread use. Hence, none of them have the computational power to fully support complex cryptographic protocols such as PIR and can only support the most basic of cryptographic primitives such as encryption, signatures, hash functions, and pseudo-random number generation, besides lacking any application-specific support (e.g., for LBS). Prior research has also shown that they need significant extensions in order to realize cryptographic protocols similar to PIR, such as Oblivious Transfer and Secure Function Evaluation [11], [25].

Designing a hardware-assisted PIR protocol wholly supported by cost-effective tokens would require significant extensions to their specifications, and would be speculative at best, and impossible at worst.

In conclusion, cryptographic protocols such as PIR and Oblivious Transfer (OT) seem to be the best solution for realizing privacy in location-based services, which don’t suffer from the drawbacks of previously proposed solutions such as $k$-anonymity or having a trusted perturbation anonymizer. There are two ways one can realize PIR for use in applications such as LBS: computational PIR and hardware-assisted PIR. We believe that hardware-assisted PIR, although a promising concept, currently requires systems to be enhanced with expensive, powerful co-processors and cannot be realized by using cost-effective, commodity hardware chips such as the TPM [9] or smartcards [20], [14], [13]. Hence we have based our methodology on computational PIR, which offers reasonable performance with no additional system setup requirements.

2.1 Our Contributions and Outline

We present an efficient solution for the exact nearest neighbour query where a location-based server does not gain any information about a querying client’s location and the client does not gain information about any location in the server’s database other than its own. Our solution is a general-purpose one and can use either a two-phase PIR or a combination of PIR and Oblivious Transfer (OT). We use the layout of the server database similar to Ghinita et al. [6] where the server super-imposes a grid on the POIs. A user $u$ in a grid cell wants to know its nearest neighbour but does not want to release its own location or the grid cell that (s)he is in.

Using the single PIR approach proposed by Ghinita et al. [6] on the grid will result in the server returning an entire column of the grid, thus releasing more information than is necessary to the client. We note that if the data (locations) are not arranged in a grid, but as a one-dimensional list, we can still use PIR to return just one cell, but this approach would incur a communication cost of $\Theta(fn)$ where $n$ is the number of grid cells and $f$ is the number of bits in the PIR modulus. Using 1-of-$n$ OT [23] by itself will require the server to transmit the whole grid of encrypted POIs to the client, the communication cost of which is $\Theta(gn)$ where $g$ is the length of the symmetric keys used in OT.

One possible solution to this problem is to use a two-phase protocol wherein the server recursively performs a PIR or OT on the grid, and the other solution is to define a two-phase protocol that combines the features of PIR and OT which brings the cost down from $\Theta(fn)$ or $\Theta(gn)$ to $\Theta(f\sqrt{n})$ and $\Theta(g\sqrt{n})$ respectively. We explore both of these approaches in our methodology. The paper is organized as follows: In Section 3 we describe the PIR and OT protocols. In Section 4 we present a two-phase framework in which one can use three different combinations of PIR and OT (PIR+OT, OT+PIR, PIR+PIR), and discuss why a two-phase OT approach (OT+OT), while possible in principle, isn’t very efficient in terms of cost. In Section 5 we describe a single-phase approach where one can use either (1) PIR over a square grid as described in [6], or (2) 1-of-$n$ OT over the entire database organized as single-dimensional list, or (3) a random OT protocol which offers a lower degree of privacy. In Section 6, we formally define the security properties of our protocol and give a proof sketch. In Section 7 we analyze the cost of our two-phase framework and compare it with the single-phase approaches.

3 Preliminaries

In this section, we give a system architecture of our proposed schemes and a table of notations that summarizes symbols that appear frequently in our protocols presented later in Section 4. We also give a brief introduction to PIR and OT, the two cryptographic protocols which are used as building blocks in our framework. Readers familiar with them can skip to Section 4.

3.1 System Architecture and Notations

Figure 1 summarizes our system architecture. The system setup does not require any third party and keys aren’t shared between users. Consider a database server which holds an $n$-element string, $X_1, \ldots, X_n$ arranged as a $\sqrt{n} \times \sqrt{n}$ matrix, and a user who wants to retrieve one element, $X_{i,j}$ from the matrix. In the theoretical definitions of PIR and OT, each element is a single bit, although in applications this is usually generalized to $m$-bit strings (as in our protocols in Section 4). The user first sends an obfuscated request, $q_{i,j} = E_2(E_1(X_{i,j}))$ using some combination of PIR and OT represented by $E_1$ and $E_2$. The server responds with a tuple $(X_{i,j}, q_{i,j})$, using which the user can later compute the value of $X_{i,j} = E_2(E_1(q_{i,j}))$. PIR and OT rely on the fact that it is computationally infeasible in polynomial time for the server to deduce the value of $i,j$ given $q_{i,j}$.

Table 1 introduces notations that are frequently referred to in Section 4.
Fig. 1. System architecture

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>Length of PIR modulus in bits</td>
</tr>
<tr>
<td>$g$</td>
<td>Length of OT symmetric keys in bits</td>
</tr>
<tr>
<td>$M$</td>
<td>Location database matrix maintained by server</td>
</tr>
<tr>
<td>$n$</td>
<td>Total number of cells in server’s database, $M$</td>
</tr>
<tr>
<td>$\sqrt{n}$</td>
<td>Number of cells in each column</td>
</tr>
<tr>
<td>$m$</td>
<td>Number of bits in each cell</td>
</tr>
<tr>
<td>$M_{\text{opt}}$</td>
<td>Cell querying user is located in</td>
</tr>
<tr>
<td>$p_1, \ldots, p_{\sqrt{n}}$</td>
<td>Granularity of matrix</td>
</tr>
<tr>
<td>$p, p'$</td>
<td>Large primes of length $f/2$ bits</td>
</tr>
<tr>
<td>$N = p \cdot p'$</td>
<td>Composite modulus of length $f$ bits</td>
</tr>
<tr>
<td>$\oplus$</td>
<td>Binary XOR operation</td>
</tr>
</tbody>
</table>

TABLE 1
Description of Notations

3.2 Private Information Retrieval (PIR)

Private Information Retrieval (PIR), first introduced by Chor et al. [2], is a technique or protocol to let a user retrieve information from a database server without the database server knowing the element or record that has been retrieved. In the traditional definition of PIR (computational PIR), the database server has an $n$-bit string $X = X_1, \ldots, X_n$, and the client wants to know the value of $X_I$. The client sends a request to the server in the form of an obfuscated vector, $E(I) = q$, where $E$ denotes the algorithm used for generating the obfuscated vector. The server responds with a value $v(X, q)$. Using this, the client can compute the value of $X_I$. Typically we require that the client’s request remain private in the presence of a computationally-bounded adversary, which is referred to as “computational PIR” (as opposed to “information theoretic PIR” in which the adversary is not bounded). Kushilevitz and Ostrovsky’s well-known solution [19] to the computational PIR problem is based on the computational intractability of deciding whether a given number is a quadratic residue of a composite modulus. Although there are other forms of PIR such as hardware-assisted PIR, multiple-server PIR and symmetric PIR, in this paper we focus on computational PIR and the protocol from Kushilevitz and Ostrovsky. In the following paragraph we give a high-level description of this solution to the computational PIR problem.

Let $p$ and $p'$ be large primes, and $N = p \cdot p'$ be a composite modulus. Let $\mathbb{Z}_N^*$ denote the set of numbers that are co-prime or relatively prime with $N$. The set of quadratic residues is defined by $QR_N = \{y \in \mathbb{Z}_N^* | \exists x \in \mathbb{Z}_N^* : y = x^2 \mod N\}$, and the set of non-quadratic residues is the complement of $QR_N$, written $QR_N^c$. If we denote the set $\mathbb{Z}_N^*$ by $\{y/N \} = Y/N$, then $Y/N$ denotes the Jacobi symbol, exactly half of the numbers in $\mathbb{Z}_N^*$ are in the set $QR_N$ and the other half are in $QR_N^c$. Determining whether a given $x \in \mathbb{Z}_N^+$ is in $QR_N$ or in $QR_N^c$ is easy if the factorization of $N$ is known, but is commonly believed to be computationally intractable if the factorization of $N$ is unknown; this is a widely-used cryptographic assumption known as the “quadratic residuosity assumption.” For a detailed theoretical exposition, readers are referred to [2].

To form a PIR query, the client first randomly selects two $f/2$ bit primes, $p$ and $p'$ and computes $N = p \cdot p'$ where $N$ is $f$ bits long. The query is then formed as $E(I) = q = [q_1, \ldots, q_n]$, where each $q_i$ is drawn from $Z_N^+$ such that $q_i \in QR_N^c$ and $\forall i \neq I, q_i \in QR_N$. The server computes $v(X, q) = \Pi_{j=1}^{n} w_j$, where $w_j = q_j^2$ if $X_j = 0$, or $q_j$ otherwise. Note that $v(X, q) \in QR_N$ if and only if $X_I = 0$, so the client uses its knowledge of $p$ and $p'$ to test whether $v(X, q) \in QR_N$ and thus determines the value of $X_I$. As just described, this produces a query of size $\Theta(fn)$ and a response of size $\Theta(f)$; however, in the simplest form of the Kushilevitz/Ostrovsky protocol the server’s data is organized into a $\sqrt{n} \times \sqrt{n}$ matrix, and this basic PIR step is run on every row with a single $E(I)$ vector. As a result, the query size is $\Theta(\sqrt{fn})$ and the response size is also $\Theta(\sqrt{fn})$. Further optimizations are possible — see the original paper for details [19].

We note that, like many cryptographic protocols, the traditional PIR definition above is one in which data items are single bits. In practice the items would typically be multiple bits, which is easily accomplished by running multiple instances of the PIR protocol in parallel.

3.3 Oblivious Transfer (OT)

Oblivious Transfer is a cryptographic protocol that was introduced by Brassard, Crépeau and Roberts [1] and is used as a fundamental construct in various cryptographic protocols, including those for multi-party secure function evaluation. Consider a setting where Bob has a string of bits $X_1, \ldots, X_n$ and Alice wants to know one of them. Alice does not want Bob to know which bit she has chosen, and Bob does not want Alice to know any bit other than the one she has chosen. The solution for this is known as the 1-of-$n$ OT protocol. There are two variants of this protocol: the 1-of-2 OT protocol and the $k$-of-$n$ OT protocol. In 1-of-2 OT, Bob has 2 elements, $X_0$ and $X_1$. Alice chooses to know one of them. Alice receives exactly one element without learning anything about the other element, while Bob remains oblivious as to which element was sent. In the $k$-of-$n$ OT protocol, Bob has a string of bits $X_1, \ldots, X_n$, Alice wants to know a subset of $k$ of these bits. Many efficient protocols have been proposed for the Oblivious Transfer problem, including a protocol in the standard model due to Naor and Pinkas [23], and more recently a hardware-assisted protocol due to Gunupudi and Tate [11]. In this paper we use Naor and Pinkas’s OT [23] — we describe their technique for doing 1-of-$N$ OT using an existing 1-of-2 OT below. The protocols below handle $m$-bit messages directly, rather than relying on multiple single-bit OTs.

Let Bob be a server who holds an $N$-element string $X_1, \ldots, X_N$ where each $X_i \in \{0, 1\}^m$ is an $m$-bit value. Let Alice be a client who wishes to obtain element $X_I$. The 1-of-$N$ OT protocol proceeds thus: Bob first prepares $l = \lceil \log N \rceil$
random key-pairs \((K_0^b, K_1^b), (K_2^b, K_3^b), \ldots, (K_{l-1}^b, K_l^b)\) where
for all \(1 \leq j \leq l\) and \(b \in \{0, 1\}\), \(K_j^b\) is a \(t\)-bit key to a pseudo-
random function \(F_K\). For all \(1 \leq I \leq N\) let \((i_1, i_2, \ldots, i_j)\)
be the bits of \(I\). Bob encryps each element in the string
as \(Y_I = X_I \oplus \bigoplus_{j=1}^{l} F_{K_i^b}(I)\) and sends all the encrypted
strings \(Y_1, \ldots, Y_N\) to Alice. Alice and Bob then engage in
a 1-of-2 OT for each pair of keys \((K_0^b, K_1^b)\). If Alice wants
element \(X_I\), she picks key \(K_i^b\). Finally Alice reconstructs
\(X_I = Y_I \oplus \bigoplus_{j=1}^{l} F_{K_i^b}(I)\).

\section{A Two-phase Framework}

For organizing the location database, we follow the same basic methodology given in [6]. As part of pre-processing, the whole
space is divided into Voronoi tessellations using the set of
POIs, \(S = \{s_1, s_2, \ldots, s_n\}\) and a grid or matrix of size \(M = \lceil \sqrt{n} \rceil \times \lceil \sqrt{n} \rceil\) is superimposed on top of it as shown in Figure 2.
For each grid cell \(gc\) (which can also be referred to as \(M_{a,b}\)
where \(a\) is the row number and \(b\) is the column number), the
server computes the POIs located in all the Voronoi cells that
intersect \(gc\) and prepares a neighbour list that contains the
possible nearest neighbours for any point in the cell. Figure
2 shows the neighbour list for each grid cell. The server sets
the maximum number of POIs in the lists to a constant fixed number, \(P_{\text{max}}\) (e.g., in Figure 2, \(P_{\text{max}} = 3\)). In case some of
the cells do not have enough POIs in the list, the server pads
the list with duplicate entries till it reaches the value of \(P_{\text{max}}\).
This can be seen in Figure 2 in the neighbour lists for \(A_1\), \(C_2\), etc. In the figure, we have used duplicate entries for the
padding, but one can use a pre-agreed padding value as well.

When a user \(u\) with location \(l\) requests its nearest neighbour,
it simply requests the neighbour list associated with the grid
cell that contains \(l\) and filters the results to find the nearest
neighbour of \(l\). Let \(M_{a,b}\) be the matrix cell corresponding to
the user’s location \(l\). Using PIR, in [6], the user selects a query
message \(y = [y_1, \ldots, y_{\lceil \sqrt{n} \rceil}]\) \((y_i \in \mathbb{Z}^+_{\sqrt{n}}, i = 1, 2, \ldots, \sqrt{n})\)
where \(y_b \in QR^r\) and \(v_j \neq b, y_{j} \in QR\) (Here \(b\) is the column
the user is located in). The server computes for each row, \(r\), of
the matrix, \(M\), the value \(v_r = \Pi_{j=1}^{n} (w_{r,j})\) and returns \(z = [z_1, \ldots, z_{\lceil \sqrt{n} \rceil}]\). Here \(w_{r,j} = y_{j}^r\) if \(M_{r,j} > 0\), or \(y_{j}\)
otherwise. The client can then use a similar method as introduced in
Section 3 to calculate the values of \(M_{a,b}\) where \(r\) represents
any row in \(M\).

Since the data (locations) are organized in a grid, this
leads to the server returning extra information which the user
has not queried about, wherein most of the data returned
is not relevant. Specifically, if the user requests one grid or
matrix cell \(M_{a,b}\), the server, along with \(M_{a,b}\) returns the
contents of the entire column \(b\). Since the size of the data
returned increases with the size of the grid, the user has
to sift through a lot of extraneous data, or data that might
not be particularly useful to get their nearest neighbours.
In the optimization techniques proposed in [6], data compression
and using a rectangular matrix instead of a square one are
considered as ways to reduce the amount of data returned by
the server. By compressing the query results returned by the
server, we would just be removing duplicates and reducing the

\section{Protocols}

\subsection{Exact NN with PIR+OT}

1. \textbf{Server} – Bob uses the POIs \(S\) to create Voronoi tessellations of the space.
2. Bob divides the space into an \(\sqrt{n} \times \sqrt{n}\) grid \(M\), so
the grid is represented by \((M_{1,1}, \ldots, M_{\sqrt{n}, \sqrt{n}})\). The
neighbour list for each grid cell is prepared. Here, the
contents of each grid cell is an \(m\)-bit string.
3. \textbf{User} – Alice initiates a query in order to find the
neighbour list of the cell she is in. Alice also randomly
generates a composite modulus \(N = p \cdot p'\), where \(p, p'\)
are large primes, and sends \(N\) to Bob. Note that only
Alice knows the factorization of \(N\), not Bob.
4. Bob sends the granularity of the grid.
5. Let Alice be located in \(M_{a,b}\). Alice selects the column
in which her cell is located, i.e., \(\text{column } b\). At this point,
Alice can choose to engage in either of two protocols:
PIR or OT_{\sqrt{n}}. For this protocol, Alice chooses
to engage in PIR.
Let $PIR(b) = [y_1, \ldots, y_{\sqrt{n}}]$ such each $y_i \in Z_N^{+1}$, $y_b \in QR_N$, and $\forall j \neq b, y_j \notin QR_N$. Alice issues a PIR request $PIR(b)$ to Bob.

6) Bob prepares the PIR response, $z = [z_1, \ldots, z_{\sqrt{n}}]$ for each row, $r$, of the matrix thus:

$$z_r = \prod_{j=1}^{\sqrt{n}} (w_{r,j})$$

where $w_{r,j} = y_j^2$ if $Mr,j = 0$, or $y_j$ otherwise. In total there will be $m$ responses for each $z_i$ (denoted by $z_i[1], z_i[2], \ldots, z_i[m]$). At this point Bob can choose to engage in either of two protocols: $OT^{\sqrt{n}}_{SERVER}$ or $PIR$. For this protocol, Bob chooses $OT^{\sqrt{n}}_{SERVER}$. Bob runs the protocol $OT^{\sqrt{n}}_{SERVER}(z)$ where $z$ is the PIR response for each bit of the $m$-bit string.

7) Alice runs the $OT^{\sqrt{n}}_{CLIENT}$ protocol for obtaining the $a^{th}$ element in $z$. In the course of the $OT^{\sqrt{n}}_{CLIENT}(a,Y)$ protocol, Alice gets the encrypted values $Y = [Y_1, \ldots, Y_{\sqrt{n}}]$, of which Alice can only decrypt $Y_a = z_a$. After decrypting $z_a$ Alice can check if each bit $i$ ($i = 1, 2, \ldots, m$) of $z_a[i]$ is in $QR_N$ or $z_a[i] \in QR_N$ using the formula $(z_a[i])^{2A} = 1 \mod p$ \quad $\land$ \quad $(z_a[i])^{2A} = 1 \mod p'$. If this expression evaluates to true, then it implies that $z_a[i]$ is in $QR_N$. For each bit, if $z_a[i] \in QR_N$, then Alice concludes that bit is 0, else if $z_a \in QR_N$, the bit is 1. Alice repeats this process for each bit of the $m$-bit string. Once she does this, she can get the contents of her cell $M_a,b$. From the contents of $M_a,b$, she can get the list of neighbours (POIs) associated with $M_a,b$. Once Alice knows the set of POI’s, she can calculate their individual distances and find her nearest neighbour.

8) Return the nearest neighbour.

**Protocol 2 Exact_NN with OT+PIR**

1) Server – Bob uses the POIs $S$ to create Voronoi tessellations of the space.

2) Bob divides the space into an $\sqrt{n} \times \sqrt{n}$ grid $M$, so the grid is represented by $(M_{1,1}, \ldots, M_{\sqrt{n},\sqrt{n}})$; The neighbour list for each grid cell is prepared. Bob converts each column (along with its cells’ neighbours) into a string: $(X_1 = [X_{1,1}, \ldots, X_{1,\sqrt{n}}], \ldots, X_{\sqrt{n}} = [X_{\sqrt{n},1}, \ldots, X_{\sqrt{n},\sqrt{n}}])$, with each element $m$ bits long. So in all there will be $\sqrt{n}$ strings. Bob then executes the $OT^{\sqrt{n}}_{SERVER}(X)$ on each column of the grid and creates $\sqrt{n}$ encrypted columns: $(Y_1 = [Y_{1,1}, \ldots, Y_{1,\sqrt{n}}], \ldots, Y_{\sqrt{n}} = [Y_{\sqrt{n},1}, \ldots, Y_{\sqrt{n},\sqrt{n}}])$. Please note that Bob needs to encrypt each column of the grid with a different set of keys, but the same key is used for encrypting all the cells in a single column. So Bob needs to generate a total of $\log \sqrt{n}$ keys, one for each column. Bob does not engage in the 1-of-2 OT with Alice at this point to give her the keys, but waits until after the PIR step.

3) User – Alice initiates a query in order to find the neighbour list of the cell she is in (for each bit in the neighbour list). Alice also randomly generates a composite modulus $N = p \cdot p'$ and sends $N$ to Bob ($p,p'$ are large primes). Note that only Alice knows the factorization of $N$.

4) Bob sends the granularity of the grid.

5) Alice chooses the encrypted column containing her cell $M_{a,b}$, e.g., column $Y_b = [Y_{1,b}, \ldots, Y_{\sqrt{n},b}]$, and engages in PIR. Let $Z_N$ denote the set of integers that are co-prime with $N$. Denote the set of quadratic residues (QR’s) modulo $N$ using the formula:

$$QR_N = \{y \in Z_N | \exists x \in Z_N^* : y = x^2 \mod N\}$$

Let $PIR(b) = [y_1, \ldots, y_{\sqrt{n}}]$ such each $y_b \in QR_N$ and $\forall j \neq b, y_j \notin QR_N$. Alice issues a PIR request $PIR(b)$ to Bob.

6) Bob prepares the PIR response: $z = [z_1, \ldots, z_{\sqrt{n}}]$ for each row, $r$, of the column $b$ thus:

$$z_r = \prod_{j=1}^{\sqrt{n}} (w_{r,j})$$

where $w_{r,j} = y_j^2$ if $Mr,j = 0$, or $y_j$ otherwise. Please note that the PIR request and response are prepared over Bob’s encrypted database as opposed to the plaintext database as in the Exact_NN with PIR+OT and Exact_NN with PIR+PIR protocols. Bob sends the $z$ vector to Alice.

7) For each element of the encrypted $z$ vector, Alice checks if each bit $i$ ($i = 1, 2, \ldots, m$) of $z_a[i]$ is in $QR_N$ or $z_a[i] \in QR_N$ using the formula $(z_a[i])^{2A} = 1 \mod p$ \quad $\land$ \quad $(z_a[i])^{2A} = 1 \mod p'$. If this expression evaluates to true, then it implies that $z_a[i]$ is in $QR_N$. For each bit, if $z_a[i] \in QR_N$, then Alice concludes that bit is 0, else if $z_a \in QR_N$, the bit is 1. Alice repeats this process for each bit of the $m$-bit string. Once she does this, she can get the contents of her cell $M_{a,b}$. From the contents of $M_{a,b}$, she can get the list of neighbours (POIs) associated with $M_{a,b}$. Once Alice knows the set of POI’s, she can calculate their individual distances and find her nearest neighbour.

8) Return the nearest neighbour.

**Protocol 3 Exact_NN with PIR+PIR**

1) Server – Bob uses the POIs $S$ to create Voronoi tessellations of the space.

2) Bob divides the space into an $\sqrt{n} \times \sqrt{n}$ grid $M$, so the grid is represented by $(M_{1,1}, \ldots, M_{\sqrt{n},\sqrt{n}})$; The neighbour list for each grid cell is prepared. Bob converts each column (along with its cells’ neighbours) into a string: $(X_1 = [X_{1,1}, \ldots, X_{1,\sqrt{n}}], \ldots, X_{\sqrt{n}} = [X_{\sqrt{n},1}, \ldots, X_{\sqrt{n},\sqrt{n}}])$, with each element $m$ bits long. So in all there will be $\sqrt{n}$ strings. Bob then executes the $OT^{\sqrt{n}}_{SERVER}(X)$ on each column of the grid and creates $\sqrt{n}$ encrypted columns: $(Y_1 = [Y_{1,1}, \ldots, Y_{1,\sqrt{n}}], \ldots, Y_{\sqrt{n}} = [Y_{\sqrt{n},1}, \ldots, Y_{\sqrt{n},\sqrt{n}}])$. Please note that Bob needs to encrypt each column of the grid with a different set of keys, but the same key is used for encrypting all the cells in a single column. So Bob needs to generate a total of $\log \sqrt{n}$ keys, one for each column. Bob does not engage in the 1-of-2 OT with Alice at this point to give her the keys, but waits until after the PIR step.

3) User – Alice initiates a query in order to find the neighbour list of the cell she is in (for each bit in the neighbour list). Alice also randomly generates a composite modulus $N = p \cdot p'$ and sends $N$ to Bob ($p,p'$ are large primes). Note that only Alice knows the factorization of $N$. 
4) Bob sends the granularity of the grid.
5) Let Alice be located in $M_{a,b}$. Alice selects the column in which her cell is located, i.e., column $b$. At this point, Alice can choose to engage in either of two protocols: PIR or $OT_{\sqrt{n}}^{\text{CLIENT}}$. For this protocol, Alice chooses to engage in PIR. Let $Z_N^*$ denote the set of integers that are co-prime with $N$. Denote the set of quadratic residues (QR’s) modulo $N$ using the formula:

$$QR_N = \{y \in Z_N^* | \exists x \in Z_N^*: y = x^2 \mod N\}$$

Let $PIR(b) = [y_1, \ldots, y_{\sqrt{n}}]$ such that $y_b \in QR_N$ and $\forall j \neq b, y_j \notin QR_N$. Alice issues a PIR request $PIR(b)$ to Bob.

6) Bob computes for each row, $r$, of the matrix, $M$, the value $z_r = \prod_{j=1}^{\sqrt{n}}[w_{r,j}]$ where $w_{r,j} = y_j^2$ if $M_{r,j} = 0$, or $y_j$, and returns $z_r = [z_1, \ldots, z_{\sqrt{n}}]$ (please note this process needs to be done for every bit of the contents of the cell).

7) Alice sends a request $Y' = [y'_1, \ldots, y'_{\sqrt{n}}]$ with only $y'_a \in QR_{N_a}$ for each bit $i$ of the vector $z$ denoted by $z_i[a]$.

8) Bob computes the value $z' = \prod_{j=1}^{\sqrt{n}}[w_{j,i}]$ where $w_{j,i} = y_j^2$ if $z_j[i] = 0$, or $y_j$ otherwise. The server returns $z'$ to the client.

9) Alice recovers $z_a$ from $z'$ (bit by bit) and then $M_{a,b}$ from $z_a$ using a similar approach as described in the $\text{EXACT}_\text{NN}$ with PIR+OT and $\text{EXACT}_\text{NN}$ with OT+PIR. From the contents of $M_{a,b}$, she can get the list of neighbours (POIs) associated with $M_{a,b}$. Once Alice knows the set of POI’s, she can calculate their individual distances and find her nearest neighbour.

10) Return the nearest neighbour.

Sub-protocol 1 $OT_{\sqrt{n}}^{\text{SERVER}}(X)$

1) Let $X = [X_1, X_2, \cdots, X_{\sqrt{n}}]$. The client would like to know the contents of one element $X_i$. Let $l = \log_2 \sqrt{n}$.

2) The server chooses a symmetric encryption algorithm $E_K$, which uses key $K$ and generates $l$ random pairs of keys. The pairs of keys are represented by $(K^0_{i,j}, K^1_{i,j})$, $(K^0_{i,j}, K^2_{i,j}), \ldots, (K^0_{i,j}, K^l_{i,j})$. So, $K^b_{i,j}$ is a key for the encryption algorithm $E_K$ $\forall 1 \leq j \leq l$ and $\forall b \in \{0, 1\}$.

3) For each $1 \leq i \leq \sqrt{n}$, let $\langle p_1, p_2, \ldots, p_l \rangle$ represent the bits of $i$, the server prepares an encryption: $Y_i = X_i \oplus \bigoplus_{j=1}^{l}(E_{K^j_{i,j}}(i))$. Thus, there are a total of $\sqrt{n}$ encryptions.

4) For all $1 \leq j \leq l$, the server engages in a 1-of-2 OT with the client on the strings $(K^0_{i,j}, K^1_{i,j})$.

5) The server sends the strings $Y = [Y_1, \ldots, Y_{\sqrt{n}}]$ to the client.

Sub-protocol 2 $OT_{\sqrt{n}}^{\text{CLIENT}}(i, Y)$

1) The client picks an index $i$ where $\langle p_1, p_2, \ldots, p_l \rangle$ represents the bits of $i$ to get from the server.

2) The client engages in a 1-of-2 OT with the server to learn the key $K^j_{i,j}$ for all $1 \leq j \leq l$.

3) Once the client gets the value of $Y = [Y_1, \ldots, Y_{\sqrt{n}}]$ from the server, it can reconstruct $X_i$ thus: $X_i = Y_i \oplus \bigoplus_{j=1}^{l}(E_{K^j_{i,j}}(i))$.

We have given three protocols using different combinations of PIR and OT: PIR+OT, OT+PIR and PIR+PIR. A possible fourth protocol would be OT+OT. One can implement this option by first having the server encrypt the grid row-wise and then re-encrypting the encrypted grid column-wise. The server and client would then perform two $OT_{\sqrt{n}}^{\text{SERVER}}$’s for the client to get its cell’s row and column keys. The communication cost of performing OT+OT is the same as that required for a single-level OT ($OT_{\sqrt{n}}^{1}$ over the entire database), since, in single-level OT, the server and client need to perform $OT_{\sqrt{n}}^{1}$ over $\log n$ keys, whereas in OT+OT, they need to perform $OT_{\sqrt{n}}^{1}$’s over two vectors of $\log(\sqrt{n})$ key-pairs (and $\log n = 2\log(\sqrt{n})$). The computation cost on the server’s side would be a lot more in OT+OT too, since the server needs to encrypt the entire database twice. Hence the efficiency gain (computation and communication-wise) from this would not be as significant as the other 3 two-phase protocols, and the cost of performing OT+OT would be close to, if not more than single-phase OT. Hence we do not pursue the OT+OT approach any further.

We note that the PIR+PIR protocol is quite similar to the recursive PIR protocol given in Kushilevitz and Ostrovsky [19]. The major difference between their protocol and ours is that in the recursive scheme given in [19], if we assume each element of the $z = [z_1, \ldots, z_{\sqrt{n}}]$ vector to be $f$-bits long, where $f$ is the length of the PIR modulus $N$, the client and server engage in $f$ executions of the PIR scheme, once for getting each bit of the element $z_a$ ($z_a$ is the element the client wants). In our scheme, the client gets $z_a$ in one go. Additionally, the PIR+OT and OT+PIR schemes are essentially equivalent to constructing a symmetric PIR scheme where a client gets exactly one element from the server. Kushilevitz and Ostrovsky had suggested a way in which their PIR protocol can be extended to support symmetric PIR, but their solution requires the client and server to execute a zero knowledge proof in which the client proves that it followed the PIR protocol correctly. Such zero knowledge proofs are expensive protocols and should generally be avoided in practice. Using our protocols, we can implement symmetric PIR, but without having to execute a zero knowledge proof. We note that the idea of constructing a symmetric PIR scheme using PIR and 1-of-$n$ OT was also pointed out by Naor and Pinkas [23].

5 Single-phase approaches

In this section, we describe three single-phase approaches for the purpose of comparing performance with the two-phase approaches. The single-phase approaches include a single-phase PIR protocol, a single-phase 1-of-$n$ OT protocol and a “random” OT protocol. In the first case, single-phase PIR, the server database is organized as a grid in exactly the same way and the protocol is the same as given in [6]. In the second case, single-phase OT, the database is organized as a one-dimensional list and the protocol is a simple invocation of $OT_{\sqrt{n}}^{\text{SERVER}}$ and $OT_{\sqrt{n}}^{\text{CLIENT}}$. These are exactly the same as $OT_{\sqrt{n}}^{\text{SERVER}}$ and $OT_{\sqrt{n}}^{\text{CLIENT}}$ with all instances of
\(\sqrt{n}\) replaced by \(n\). In the random OT case, we organize the database as a grid and do a 1-of-\(k\) OT over \(k\) randomly chosen cells (chosen by the client). The last approach is less expensive than the two-phase approaches or other single-phase approaches but with the caveat that it isn’t completely privacy-preserving — the server has a 1/\(k\) probability of guessing the client’s location.

**Protocol 4 Exact_NN with PIR**

1. **Server -** Bob uses the POIs \(S\) to create Voronoi tessellations of the space.
2. Bob arranges the database as a single-dimensional grid with \(n\) cells, \(X_1, \ldots, X_n\). Here, the contents of each grid cell is an \(m\)-bit string.
3. **User -** Alice initiates a query in order to find the nearest neighbour list. The server has a \(1/k\) probability of guessing the cell she is in (for each bit in the neighbour list).
4. **Bob sends the value of** \(n\).
5. Let Alice be located in \(X_i\). Alice issues a PIR request \([y_1, \ldots, y_{\lceil n/2 \rceil}]\) to Bob such that \(y_i \notin QR_N\) and for all \(j \neq i\), and \(y_i \in QR_N\).
6. Bob prepares the PIR response, \(z = [z_1, \ldots, z_{n/2}]\) for each bit of the \(m\)-bit string. So there will be \(m\) responses for each \(z_i\) (denoted by \(z_i[1], z_i[2], \ldots, z_i[m]\)).
7. From the previous step, Alice gets \(z_i[1], z_i[2], \ldots, z_i[m]\), of which Alice can only decrypt \(z_a\). After decrypting \(z_a\), Alice checks if each bit \(i \in \{1, 2, \ldots, m\}\) of \(z_a[i] \in QR_N\) or \(z_a[i] \in QR_N^c\) using the formula \((z_a[i]^{k-1} = 1 \mod p)\land(z_a[i]^{k^2} = 1 \mod p')\). For each bit, if \(z_a[i] \in QR_N\), then Alice concludes that bit is 0, else if \(z_a[i] \in QR_N^c\), the bit is 1. Alice repeats this process for each bit of the \(m\)-bit string. Once she does this, she can get the contents of her cell \(X_i\). From the contents of \(X_i\), she gets the list of neighbours (POIs) associated with \(X_i\). Once Alice knows the set of POIs’, she calculates their individual distances and finds her nearest neighbour.
8. Return the nearest neighbour.

**Protocol 5 Exact_NN with OT**

1. **Server -** Bob uses the POIs \(S\) to create Voronoi tessellations of the space.
2. Bob arranges the database as a single-dimensional grid with \(n\) cells, \(X_1, \ldots, X_n\). Here, the contents of each grid cell is an \(m\)-bit string.
3. Alice calls \(OT^n_{A}\) \(\text{CLIENT}(i, Y)\), where \(i\) is the index of the cell Alice wants.
4. Bob executes \(OT^n_{A}\) \(\text{Server}(X)\) and prepares the response \(Y = [Y_1, \ldots, Y_{\lceil n/2 \rceil}]\) to return to Alice.
5. Alice computes \(X_i\) from \(Y = [Y_1, \ldots, Y_{\lceil n/2 \rceil}]\), and from the contents of \(X_i\), gets her list of neighbours and computes the nearest neighbour based on minimum individual distance.

**6 Definitions, Proof and Complexity**

In this section, we define the security properties that are required for the client and server in the two-phase framework and then show that our protocols satisfy them. PIR and OT have been individually proven secure [19], [23], and we do not reproduce the proofs here. We need to prove that our combination of PIR and OT retains the relevant security properties. Additionally, we assume that the OT protocol in [23] uses a secure 1-of-2 OT protocol. Our definitions are based on the concept of computational indistinguishability and we assume our protocols are executed in the presence of static, semi-honest adversaries. Both of these concepts are defined below as in [8].

**Computational indistinguishability:** Let there be two distribution ensembles, \(X = \{X_k\}_{k \in \mathbb{N}}\) and \(Y = \{Y_k\}_{k \in \mathbb{N}}\), where \(k\) is a security parameter. We say that \(X\) and \(Y\) are computationally indistinguishable if, for every non-uniform polynomial-time circuit family, \(\{C_k\}_{k \in \mathbb{N}}\) (also known as distinguisher) and every positive polynomial \(p(\cdot)\), it holds that:

\[
\left| \Pr[C_k(X_k) = 1] - \Pr[C_k(Y_k) = 1] \right| < \frac{1}{p(k)}
\]

**Definition 6.1 (Secure function evaluation):** Let \(f : \{(0, 1)^n\}^m \rightarrow \{(0, 1)^n\}^m\) be an \(m\)-ary functionality, where \(f_i(x_1, \ldots, x_m)\) denotes the \(i\)th element of \(f(x_1, \ldots, x_m)\). For \(I = \{i_1, \ldots, i_l\} \subseteq [m] = \{1, \ldots, m\}\), let \(f_I(x_1, \ldots, x_m)\) denote the subsequence \(f_{i_1}(x_1, \ldots, x_m), \ldots, f_{i_l}(x_1, \ldots, x_m)\). Let \(I\) be an \(m\)-party protocol for computing \(f\). The view of the \(i\)th party during an execution of \(I\) on \(\bar{x} = (x_1, \ldots, x_m)\), is denoted by \(\text{view}_{i}(\bar{x})\), and for \(I = \{i_1, \ldots, i_l\}\), we let \(\text{view}_{I}(\bar{x}) = \cap_{i \in I} \text{view}_{i}(\bar{x})\). Let \(\Xi\) denote computational indistinguishability by non-uniform families of polynomial-time circuits. We say that \(I\) privately computes \(f\) if there
exists a probabilistic polynomial-time algorithm denoted $S$, such that for every $I \subseteq [n]$, it holds that
\[ \{S(I, (x_1, c, \cdots, x_n), f_1(\tilde{x})), f(\tilde{x})\}_{\tilde{x} \in \{0,1\}^m} \equiv \{(\text{view}^I_{\Pi}(\tilde{x}), \text{output}^I_{\Pi}(\tilde{x}))\}_{\tilde{x} \in \{0,1\}^m} \]
Informally put, this says that the view of all the parties in $I$ can be efficiently simulated solely based on their inputs and outputs.

### 6.1 Definition of privacy properties

Definition 6.2 gives the formal definition of the desired privacy properties, but these properties also have simple intuitive descriptions. The Correctness property states that if the client and server are honest and the request the neighbours of a grid cell $i$, then the client will learn the neighbours of cell $i$. The Client’s Privacy property states that the server’s view of the transaction when the client requests the neighbours of cell $i$ is computationally indistinguishable from the case where the client requests the neighbours of a different cell $i’$. The Server’s Privacy property states that the client’s view of the transaction can be completely simulated by a polynomial-time simulator that is given access to just the client’s inputs and output, such that the simulated and real executions are computationally indistinguishable. This implies that a corrupted client cannot gain any information that it is not meant to learn.

#### 6.1.1 Formal definition

**Definition 6.2 (exactNN Security Properties):** As before, we let $M_{i,j}$ for $I \subseteq i,j \leq \sqrt{n}$ be the cells in the matrix $M$, where each cell contains the list of Voronoi regions that intersect the cell and is exactly $m$ bits long (with padding added if necessary). Let $k$ be a security parameter that determines, among other things, the length of the keys in the cryptographic constructs that are used in our protocols — the size of the input $(nm)$ should also be bounded by a polynomial in $k$. Let $I = (a, b)$ denote the client’s location, so the full input to the exactNN computation is $X = (I, M)$, with $I$ being known to the client and $M$ being known to the server. For an exactNN protocol $\Pi$, let $\text{view}^I_\Pi(I, M)$ and $\text{output}^I_\Pi(I, M)$ denote the server’s and client’s views of the protocol, respectively (note that since $I$ can be randomized, these are actually random variables, or ensembles indexed by $(I, M)$). Similarly, let $\text{output}^C_\Pi(I, M)$ denote the output of the client when running protocol $\Pi$ (note that while the server interacts with the client, it does not have an “output” or result of its own). Given these definitions, an exactNN protocol $\Pi$ is secure if the following properties hold:

1) Correctness: For honest client $c$ and honest server $s$ interacting as defined by $\Pi$,
\[ Pr[\text{output}^C_\Pi(I, M) \neq M_1] \leq \frac{1}{p(k)} \]
for any polynomial $p(k)$ and sufficiently large $k$.

2) Client’s Privacy: For any PPT server $s$ (not necessarily the honest server specified by protocol $\Pi$), there exists a polynomial time simulator $S_s$ such that
\[ S_s(M) \equiv \text{view}^I_\Pi(I, M). \]

Note that since the simulation $S_s(M)$ operates independently of $I$, it follows that for every $I, I' \in \{1, \ldots, \sqrt{n}\} \times \{1, \ldots, \sqrt{n}\}$,
\[ \text{view}^I_s(I, M) \equiv \text{view}^{I'}_s(I', M). \]

3) Server’s Privacy: For any PPT client $c$ (not necessarily the honest client specified by protocol $\Pi$), there exists a polynomial time simulator $S_c$ such that
\[ S_c(I, M_1) \equiv \text{view}^I_\Pi(I, M). \]

Note that $S$ is given only $c$’s input $I$, and the correct output $M_1$, so there is no information leaked about the other cells of $M$.

### 6.2 Theorem

As a starting point for our constructions, we assume that we are given secure protocols for PIR and OT — in our protocols we use the PIR protocol due to Kushilevitz and Ostrovsky (KO) [19] and the OT protocol due to Naor and Pinkas (NP) [23]. We need to prove that our combination of PIR and OT preserves the security properties of the original protocols. The two primary security proof models for cryptographic protocols are the simulation-based model and the reduction model. Sequentially composing two proofs in the simulation model is typically straightforward; however, the KO PIR protocol was previously proved secure in the reduction model. Hence, we first convert the PIR reduction proof of KO to a simulation-based proof, and then sequentially compose it with the simulation-based OT proof of Naor and Pinkas. We note here that the security of the PIR+PIR scheme follows directly from the security of the recursive PIR scheme presented in [19] and is not discussed further.

Readers can either refer to Section 3 of the KO paper for the description of the PIR scheme or refer to Section 3 of this paper. We re-write the KO proof in the simulation model below. Let Bob be the server with inputs $(M_{1,1}, \ldots, M_{\sqrt{n}, \sqrt{n}})$ arranged in a $\sqrt{n} \times \sqrt{n}$ matrix, and let Alice be a querying client who wants element $M_{a,b}$.

**Theorem 6.1:** The PIR protocol as described in the KO paper preserves the privacy of Alice against a corrupt static, semi-honest PPT server, Bob.

**Proof:** The correctness property follows directly from the description of the PIR scheme. For protecting against a corrupt Bob (or for preserving Alice’s privacy), we need to construct a simulator for Bob, $B(M_{1,1}, \ldots, M_{\sqrt{n}, \sqrt{n}})$ who can simulate Bob’s view of the protocol. B first picks two $f/2$-bit primes, multiplies them and gets a $f$-bit modulus $N$. Since $B$ knows the factorization of $N$, $B$ can easily generate a vector of fake values: $y_1, \ldots, y_\sqrt{n} \in Z_N^*$. This is not the same as Alice’s input $y_1, \ldots, y_\sqrt{n}$ in the real, non-simulated protocol, but is nevertheless computationally indistinguishable from the real $y$ values since $N$ belongs to a hard set for which quadratic residuosity predicates are hard to determine. It is easy to see that $B$ can generate the $z_1, \ldots, z_\sqrt{n}$ vector which would be indistinguishable from the real Bob’s $z$ vector. Hence $B$ can completely simulate the view of Bob.
Since this is PIR, not OT, we do not have to account for a corrupted Alice, since PIR only preserves Alice’s privacy, not Bob’s. Hence the proof.

Next follows our main theorem.

Theorem 6.2: Using the KO PIR protocol and the NP OT protocol, the PIR+OT and OT+PIR protocols given in Section 4 are secure EXACT_NN protocols as defined in Definition 6.2.

Proof: We consider the individual properties from Definition 6.2 below.

1) Correctness: Follows directly from the correctness of PIR and OT.
2) Client’s privacy: We construct a simulator for the server (Bob) $S_c(M)$ by combining Bob’s simulators for PIR and OT: $B_{PIR}$ from Theorem 6.1 and $B_{OT}$. If we combine the simulators in the order: $(B_{PIR}, B_{OT})$, i.e., $B_{PIR}$ goes first, since Alice’s contribution to the OT protocol is independent of the result of the PIR, we simply concatenate $B_{PIR}$ and $B_{OT}$ as the simulation of our PIR+OT protocol (Bob computes the $z_1, \ldots, z_{\sqrt{n}}$ vector, so it can be given as input to the $B_{OT}$ simulator). If we combine the simulators in the order: $(B_{OT}, B_{PIR})$, i.e., $B_{OT}$ goes first, $B_{OT}$’s output is $\sqrt{n}$ encrypted vectors, each corresponding to one column of the grid: $E(y_1), \ldots, E(y_{\sqrt{n}})$. The encrypted vectors are then given to the $B_{PIR}$ simulator which then simulates the rest of the PIR protocol over the encrypted matrix, instead of the plaintext matrix.

Since the outputs of the PIR and OT simulators are individually indistinguishable from the views of the real protocols, and since Alice’s contributions to the two protocols are independent, the concatenated simulated view is indistinguishable from the view of the sequentially composed real protocols.

In both cases (PIR+OT and OT+PIR) we have created a simulator for the server’s view, and therefore Alice’s privacy is preserved in the presence of a corrupted Bob.

3) Server’s privacy: For a client $c$ (i.e., Alice), we need to construct a simulator $S_c(I, M_I)$ that completely simulates Alice’s view of the transaction. We assume we have a simulator $A_{OT}$ that simulates Alice’s view of the OT protocol. We do not have a $A_{PIR}$ simulator since PIR, by itself is not secure against a malicious Alice. For (PIR+OT) where PIR is performed first, Bob performs the first half and does not return anything to the real Alice, so $S_c$ does not need to simulate anything. In the second half, $S_c$ can easily simulate Alice’s part by running the $A_{OT}$ simulator. If the OT keys are generated using a secure PRF and if a secure $OT_k^2$ exists, the output of $S_c$ will be computationally indistinguishable from the real Alice’s output.

In (OT+PIR), where OT goes first, $S_c$ does not have to do anything for OT, since Bob does not return any result in the first half to the real Alice. $S_c$ needs to simulate Alice’s view in the second half - PIR. For this, $S_c$ randomly generates a vector: $y_1', \ldots, y_{\sqrt{n}}'$ and the corresponding $z_1', \ldots, z_{\sqrt{n}}'$ vector, which will be fake strings, but indistinguishable from the real $y$ and $z$ vectors because of the quadratic residuosity assumption. Hence one can construct a simulator for Alice, and Bob’s privacy is preserved in the presence of a corrupted Alice. Hence any combination of PIR and OT are sequentially composable.

6.3 Complexity discussion

The computation and communication complexity of the single-phase and two-phase protocols are shown in Table 2, Table 3, Table 4, and Table 5. In the analysis, we will assume that the location database is organized as a $\sqrt{n} \times \sqrt{n}$ grid with the contents of each grid cell being $m$-bits long. The length of the PIR modulus is $f$ bits and the length of the keys used in OT are $g$ bits. In the following, we use polynomials $p(f)$ and $p(g)$ to denote the cost of basic computations on $f$-bit and $g$-bit values — at worst these are modular powering operations, and so $p(x) = O(x^3)$. The costs of basic operations used in the analysis are given below:

1) The computation cost of a user preparing a PIR request is $O(p(f)\sqrt{n})$. This is the cost of a user preparing the $y = [y_1, \ldots, y_{\sqrt{n}}]$ vector.
2) The server computation cost for PIR is $O(p(f)m\sqrt{n})$. This is the cost of the server computing the $z = [z_1, \ldots, z_{\sqrt{n}}]$ vector.
3) The computation cost of a user decrypting the $z$ array received from server as the PIR response is $O(p(f)m\sqrt{n})$.
4) The communication cost of PIR is $O(fm\sqrt{n})$.
5) The user computation cost for OT is $O(p(g) \log n)$ — in the last step of OT, the user needs to decrypt the server’s response $\log n$ times, and $g$ is the length of the keys used.
6) The server computation cost for OT is $O(p(g)n)$ — the server needs to encrypt the entire database, where $g$ is the length of keys being used.
7) The communication cost of OT is $O(gn)$, since the server sends the entire encrypted database over to user. Note that there is also the cost of $\log n$ 1-of-2 OT’s, but this cost of $O(g \log n)$ is dominated by the database communication cost.

Analysis: The single-phase PIR and single-phase OT analyses, given in Table 2 and Table 3 are just direct applications of the above costs. For analyzing the computation cost of random OT, we replace $n$ cells by $k$ cells. For the communication cost of random OT, the main cost is due to sending $k$ encrypted cells from the server, with other communication (initial communication of $k$ cells and the cost of doing 1-of-2 OTs) being insignificant in comparison. For the two-phase protocols, the computation and communications analysis is briefly explained below.

Computation analysis: In the two-phase PIR computation analysis, we consider the server’s computation cost for performing PIR twice and user’s side computation for preparing and decrypting the server’s PIR response. In the PIR+OT computation analysis, we consider the user’s computation cost
of preparing the PIR request, server’s cost of performing the PIR computation and encrypting the result of the first phase (PIR) using OT, besides generating the key pairs for OT. Also, we consider the user’s computation cost for decrypting the final result of the OT that the server sends to the user. For the OT+PIR computation analysis, we consider the server’s cost of encrypting the database and generating keys in the first step for OT, and the user’s cost for decrypting the final PIR response that the server sends. The computation cost of the two-phase protocols are given in Table 4.

Communication analysis: For the communication cost, in the PIR+OT protocol, we consider the cost of user sending the initial PIR request vector and the server sending back the final PIR response. For the PIR+OT, we consider the user sending the initial PIR vector and the server and user doing the 1-of-2 OT’s for the user to get their keys and the cost of the server sending the encrypted OT data to the user. For the OT+PIR analysis, we consider the cost of the user and server performing 1-of-2 OT’s for the user to pick their keys and the cost of the server sending the final PIR response to the user. The communication cost of the two-phase protocols are given in Table 5.

The main advantage of our two-phase scheme over single-phase PIR is that the amount of data revealed by the server is very low. Hence if we have a server that allows the user to query some part of the database, but does not want to reveal the entire database, it can do so more effectively with our protocol than with single-phase PIR (as described in [6] for square grids). It is possible to achieve this with PIR over a single-dimensional list or array of data (not a grid), but that would be too expensive. Additionally, this also reduces the burden on the client/user to sort through redundancies in the data the server sends to find the data element or item that they (client) are looking for. Also, one can use OT over the entire grid and achieve the same level of security, but the cost of doing this is significantly higher than the two-phase approach ($\Theta(gn)$ in single-phase OT vs. $\Theta(g\sqrt{n})$ in two-phase). Random OT has a lower cost than the single and two-phase approaches, but provides a lower degree of privacy: the server has a $1/k$ probability of guessing the client’s location.

7 Experiments

We used Java to implement the two-phase and single-phase frameworks and used the Stopwatch library to instrument timing measurements in the code; the numbers given in the experiments are all actual execution times. For measuring computation time at the server and user, we averaged the time taken over 100 queries. We needed two datasets to use in our experiments: a user location dataset and a Points of Interest (POIs) dataset for which we used a set of spatial datasets provided by Li et al. [21] which is a real dataset of a road network and POIs in California. The number of user locations in our experiments range from 0-100K. For generating user locations, we partially used the California road network dataset [21] which contains 21,048 locations and the rest were randomly generated by us. Each user location corresponds to a grid cell. The POIs in our experiments range from 0-100K for which we used the California POI dataset provided in [21] which contains 104,770 POIs. The dataset we used contains POIs from 62 categories such as school, church, airport, beach, etc. In the experiments, for computing the user’s nearest neighbour, we measured the Euclidean distance from the user locations to the POIs and considered all POIs in a single category. In particular, in the set of experiments where we keep the number of POIs constant and vary the number of grid cells, we set an upper limit on the number of POIs (1000). The 1000 POIs can be taken from any category as long as they are closest to the user’s location. This can easily be extended to a more restricted case where the user’s neighbours are from a specific category of POIs by reducing the limit on the POIs returned by the server and choosing POIs only from a specific category. For the restricted case, we estimate the costs will be less than the ones reported in this paper.

Since PIR requires large integers, we used the Java BigInteger data type in the PIR implementation. We measured the performance of the three protocols in our two-phase framework and the performance of the three protocols in the single-phase framework in terms of computational and communication cost. In our experiments, we varied the length of the modulus for PIR, $N$ from 576-1536 bits (the size of a public key) and varied the number of POIs and grid cells from 10K to 100K. In our implementation, we used Kushilevitz and Ostrovsky’s version of PIR [19] and Naor and Pinkas’s version of 1-of-n OT [23]. In the experiments, we have compared our two-phase protocols with protocols that offer a similar degree of privacy (but with different costs), and hence do not compare them with approaches such as k-anonymity or data perturbation which do not offer the same level of privacy that we are offering (total non-disclosure of user/server data). The comparison benchmarks we use are the protocol in [6] and our own implementation of Oblivious Transfer.

Firstly we compare the server’s computation time for the two-phase protocols: PIR+PIR, PIR+OT, OT+PIR and single-phase protocols: PIR (as proposed in [6]), OT, Random OT, for varying grid sizes from 10K to 100K, the results of which are shown in Figure 3 and Figure 4. For this set of experiments,
TABLE 4
Computational cost for two-phase protocols

<table>
<thead>
<tr>
<th>Party</th>
<th>PIR+PIR</th>
<th>PIR+OT</th>
<th>OT+PIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>$O(p(f)m\sqrt{n})$</td>
<td>$O(p(f)m\sqrt{n} + p(g)m\log\sqrt{n})$</td>
<td>$O(p(f)g\sqrt{n})$</td>
</tr>
<tr>
<td>Server</td>
<td>$O(p(f)mn)$</td>
<td>$O(p(f)mn + p(g)m\sqrt{n})$</td>
<td>$O(p(g)n + p(f)g\sqrt{n})$</td>
</tr>
<tr>
<td>Total time</td>
<td>$O(p(f)mn)$</td>
<td>$O(p(f)mn + p(g)m\sqrt{n})$</td>
<td>$O(p(g)n + p(f)g\sqrt{n})$</td>
</tr>
</tbody>
</table>

we kept the number of POIs per cell constant at 1000 (since we were varying the grid size). In the random OT protocol, we varied the number of random cells chosen by the user from 2K for 10K grid to 20K for 100K grid. We can see that PIR+PIR and PIR+OT have similar times while OT+PIR requires significantly more time. This is due to the fact that the server needs to encrypt the entire database column-by-column in OT+PIR, whereas in PIR+OT, the server just has to encrypt one column. The two-level protocols are expectedly more expensive than the single-level PIR and random OT but also offer more privacy. In particular, the client gets only their cell and its neighbors in the two-level protocols, but using single-level PIR, it would get the entire grid column, and using random OT, the server would have a $1/k$ probability of guessing the client’s location, where $k$ is the number of cells the client chooses. The two-phase protocols provide a higher level of privacy at the expense of increasing the computation time on the server’s side. The cost of single-phase OT is much higher than the two-phase protocols though, which confirms our hypothesis that a 1-of-$n$ OT over the entire grid would be too expensive, although it provides the same level of privacy.

![Fig. 3. Server computation time with varying grid size in two-phase protocols](image1)

![Fig. 4. Server computation time with varying grid size in single-phase protocols](image2)

We next compared the computation time required at the server for two-phase protocols with the single-phase protocols with varying modulus sizes. The modulus $N$ as used in PIR is the size of a public key (between 576-1536 bits). As the modulus grows it becomes harder to factor, and hence provides higher security, but incurs more computation cost. Figure 5 shows that even for a modulus of size 1536 bits, which is reasonably large, the computation time taken is a little over a minute in the two-phase protocols. We estimate that in applications such as LBS, the typical size of the modulus would be around 768 bits. Figure 6 shows the computation time for single-phase PIR which is less than the time for the two-phase protocols, since we perform computations on the modulus just once. Also, the modulus is used only in PIR, hence we haven’t shown graphs for single-phase OT and random OT. In the two-phase protocols, this is reflected in the fact that the time taken for PIR+OT and OT+PIR do not increase as much as PIR+PIR.

Figure 7 shows the communication cost in Kb for the two-phase protocols with the number of POIs returned increasing linearly from 10K-100K. The communication cost here denotes the amount of data returned from server to client. The communication cost for PIR+PIR is the least since the server just has to send the PIR response vector ($z$ vector) back to the client. For PIR+OT and OT+PIR, it is slightly higher since the server needs to send an encrypted column back to the user and also perform a 1-of-2 OT for exchanging keys. Figure 8 shows the communication cost in Mb for POIs varying from 10K to 100K for the single-phase protocols. We note that the communication cost for the single-phase protocols is in terms of Mb rather than Kb as in the two-phase protocols and is one of the major points of difference between the two-phase and single-phase protocols. In single-phase PIR, the communication cost is obviously higher since the server has to send extraneous data to the client. In single-phase OT, we do a 1-of-$n$ OT over the entire database, hence the server and client have to perform $\log n$ 1-of-2 OT’s for the client to get
its keys, besides the server having to send the entire encrypted database to the client. In the two-phase protocols where we use 1-of-\(n\) OT, the OT is performed over a single column and the amount of encrypted data and keys the server has to send to the client is much less than in OT over the entire database. In random OT, the server has to send \(k\) encrypted cells of the grid to the client and keys for decrypting a single cell out of \(k\) cells.

We next compare the user computation time with increasing number of POIs (10K-100K) in the two-phase and single-phase protocols. We can see from Figure 9 that the user computation time for PIR+OT and OT+PIR is slightly higher than PIR+PIR since the user needs to decrypt its grid cell and the cell’s nearest neighbours from the data items returned by the server using the keys obtained from the server through 1-of-2 OT. In case of PIR+PIR, the user does not have to perform any decryptions: it just needs to check whether the bits of the string returned by the server are quadratic residues or not. In the single-phase protocols in Figure 10, the user computation time is a bit higher for single-phase PIR than for two-phase PIR since the user has to perform the quadratic residue/nonresidue check for a few more cells than in the two-level case. The time taken for single-phase OT is significantly higher than the two-phase protocols since the user needs to perform decryptions over \(\log n\) elements as opposed to \(\log \sqrt{n}\) in the two-phase protocols that involve OT. The user computation time in random OT is less than the single-phase OT, but this really depends on the choice of \(k\) in the random OT. If \(k\) is too small, the cost will be less, but the privacy offered will also be lower. The largest value of \(k\) is \(n\); if we set \(k = n\), the cost will be the same as single-phase OT. This confirms that a
single-level OT, while offering the same level of privacy as the two-level protocols is much more expensive than the two-level approaches.

Figure 9 shows the user computation time with varying number of POIs in two-phase protocols. Figure 10 shows the user computation time with varying number of POIs in single-phase protocols.

Figure 11 shows the user computation time with a varying modulus, $N$, from 576 bits to 1536 bits. This is relevant only in the case of PIR+PIR since OT does not use a modulus. In the two-phase protocols, PIR+PIR shows significant variation while the PIR+OT and OT+PIR times are almost constant. Figure 12 shows the user computation time in the single-phase protocols with varying modulus bits from 576-1536 bits; for two-phase PIR, this is 2-3 seconds less than the single-phase PIR.

Figure 13 and Figure 14 show the user computation time for the two-phase protocols and single-phase protocols respectively with varying size of OT keys. The keys used in OT are symmetric keys with size ranging from 80-256 bits. We can see that in the two-phase protocols, the time for PIR+OT and OT+PIR is almost same ranging from 1-13 seconds. For single-phase OT, the user computation time goes from 1-25 seconds. Needless to say, this is due to the extra decryptions performed on the user’s side ($\log n$ decryptions in single-phase as compared to $\log \sqrt{n}$ in the two-phase approaches). We measured only the user’s computation time on varying the key size, since in typical LBS scenarios; the user’s device would be a mobile phone, PDA or any resource-constrained device while the server would be a more powerful machine. Hence one would be more interested in minimizing the computation cost on the user’s side rather than server’s side.

The three two-phase protocols: PIR+PIR, PIR+OT and OT+PIR have a slightly higher (30-50 seconds) server com-
In this paper, we have proposed a way to achieve user privacy in location-based services, while ensuring the user doesn’t learn information about any location in the server’s database other than its own. We have also defined the privacy properties desired of such protocols in general and provided a proof sketch for our protocols. Our experiments show that the two-phase protocols using PIR and OT have reasonable costs wherein most operations take around a minute and are thus feasible to use in real-world LBS applications.

One way our framework can be improved is by using a more efficient OT protocol such as the one presented by Gunupudi and Tate [11] which makes the OT protocol non-interactive as opposed to the interactive OT used in this paper. Using a non-interactive OT protocol would significantly decrease the communication cost of the OT operation; but requires extra hardware security assumptions in the form of a Trusted Platform Module chip [9] (TPM) on the server’s side. In this paper we do not consider the hardware-assisted model of computation and hence have not used [11]. Previous work has explored the idea of hardware-assisted PIR by requiring the system to be augmented with expensive, computationally powerful secure co-processors. It would be interesting to explore the idea of realizing hardware-assisted PIR using cost-effective (but computationally weak) commodity hardware chips such as the TPM or smartcards [20], [14], [13]. One can also extend our framework beyond nearest neighbour queries such as \(k\)-nearest neighbour or other spatial queries. One of the directions of future work which we are currently exploring is considering different models of communication in LBS such as the peer-to-peer model and investigate the application of cryptographic protocols to provide privacy solutions for spatial queries in them such as group nearest neighbour queries.

8 DISCUSSION AND FUTURE WORK

In this paper, we have proposed a way to achieve user privacy in location-based services, while ensuring the user doesn’t learn information about any location in the server’s database other than its own. We have also defined the privacy properties desired of such protocols in general and provided a proof sketch for our protocols. Our experiments show that the two-phase protocols using PIR and OT have reasonable costs wherein most operations take around a minute and are thus feasible to use in real-world LBS applications.

One way our framework can be improved is by using a more efficient OT protocol such as the one presented by Gunupudi and Tate [11] which makes the OT protocol non-interactive as opposed to the interactive OT used in this paper. Using a non-interactive OT protocol would significantly decrease the communication cost of the OT operation; but requires extra hardware security assumptions in the form of a Trusted Platform Module chip [9] (TPM) on the server’s side. In this paper we do not consider the hardware-assisted model of computation and hence have not used [11]. Previous work has explored the idea of hardware-assisted PIR by requiring the system to be augmented with expensive, computationally powerful secure co-processors. It would be interesting to explore the idea of realizing hardware-assisted PIR using cost-effective (but computationally weak) commodity hardware chips such as the TPM or smartcards [20], [14], [13]. One can also extend our framework beyond nearest neighbour queries such as \(k\)-nearest neighbour or other spatial queries. One of the directions of future work which we are currently exploring is considering different models of communication in LBS such as the peer-to-peer model and investigate the application of cryptographic protocols to provide privacy solutions for spatial queries in them such as group nearest neighbour queries.

8 DISCUSSION AND FUTURE WORK

In this paper, we have proposed a way to achieve user privacy in location-based services, while ensuring the user doesn’t learn information about any location in the server’s database other than its own. We have also defined the privacy properties desired of such protocols in general and provided a proof sketch for our protocols. Our experiments show that the two-phase protocols using PIR and OT have reasonable costs wherein most operations take around a minute and are thus feasible to use in real-world LBS applications.

One way our framework can be improved is by using a more efficient OT protocol such as the one presented by Gunupudi and Tate [11] which makes the OT protocol non-interactive as opposed to the interactive OT used in this paper. Using a non-interactive OT protocol would significantly decrease the communication cost of the OT operation; but requires extra hardware security assumptions in the form of a Trusted Platform Module chip [9] (TPM) on the server’s side. In this paper we do not consider the hardware-assisted model of computation and hence have not used [11]. Previous work has explored the idea of hardware-assisted PIR by requiring the system to be augmented with expensive, computationally powerful secure co-processors. It would be interesting to explore the idea of realizing hardware-assisted PIR using cost-effective (but computationally weak) commodity hardware chips such as the TPM or smartcards [20], [14], [13]. One can also extend our framework beyond nearest neighbour queries such as \(k\)-nearest neighbour or other spatial queries. One of the directions of future work which we are currently exploring is considering different models of communication in LBS such as the peer-to-peer model and investigate the application of cryptographic protocols to provide privacy solutions for spatial queries in them such as group nearest neighbour queries.

8 DISCUSSION AND FUTURE WORK

In this paper, we have proposed a way to achieve user privacy in location-based services, while ensuring the user doesn’t learn information about any location in the server’s database other than its own. We have also defined the privacy properties desired of such protocols in general and provided a proof sketch for our protocols. Our experiments show that the two-phase protocols using PIR and OT have reasonable costs wherein most operations take around a minute and are thus feasible to use in real-world LBS applications.

One way our framework can be improved is by using a more efficient OT protocol such as the one presented by Gunupudi and Tate [11] which makes the OT protocol non-interactive as opposed to the interactive OT used in this paper. Using a non-interactive OT protocol would significantly decrease the communication cost of the OT operation; but requires extra hardware security assumptions in the form of a Trusted Platform Module chip [9] (TPM) on the server’s side. In this paper we do not consider the hardware-assisted model of computation and hence have not used [11]. Previous work has explored the idea of hardware-assisted PIR by requiring the system to be augmented with expensive, computationally powerful secure co-processors. It would be interesting to explore the idea of realizing hardware-assisted PIR using cost-effective (but computationally weak) commodity hardware chips such as the TPM or smartcards [20], [14], [13]. One can also extend our framework beyond nearest neighbour queries such as \(k\)-nearest neighbour or other spatial queries. One of the directions of future work which we are currently exploring is considering different models of communication in LBS such as the peer-to-peer model and investigate the application of cryptographic protocols to provide privacy solutions for spatial queries in them such as group nearest neighbour queries.

REFERENCES


